DATA STRUCTURES: CHAPTER 1 ARRAYS AND STRINGS

#in stringbuilder.py

First attempt - Hints and comments - Second attempt - Solution (book)

Is Unique: Implement an algorithm to determine if a string has all unique characters.

```
def isUnique():
    list_char_used = []
    for i in string:
       if i in list_char_used:
         return False
           else:
         list_char_used.append(i)
    return True
```

What if you cannot use additional data structures?

```
#Compare a char one by one to the rest of the string
# if string is of length 5:
# compare 0 to 1, 2, 3, 4 (n-1)
# compare 1 to 2, 3, 4,    (n-2)...
# O(½(3n-n^2))
```

```
def isUnique():
    i=0
    while i!=(len(string)):
       for k in range(i+1, len(string)):
           if string[i]== string[k]:
               return False
        i+=1
    return True
```

checkPermutation: Given two strings, write a method to decide if one is a permutation of the other.

```
#length is the same
#all letters are used exactly the same # of times

   def checkPermutation(str1, str2):
       If len(str1)==len(str2):
          Str2list = list(str2)
          for char in str1:
               str2list.remove(char) #removes single instance
          if len(str2list) == 0:
               Return True
       return False
```

Urlify Write a method to replace all spaces in a string with '%20'.

```
#in Python, we can use replace(old, new)

     Def urlify(string):
          #requires re-assignment
          output=string.replace(“ “, “%20”)
          return output

#define without replace
     def urlify(string):
          output=""
          str_list = list(string)
          for i in range(len(string)):
               if string[i] == " ":
                  str_list[i] = "%20"
          output = output.join(str_list)#requires re-assignment
      return output
```

Palindrome Permutation: Given a string, write a function to check if it is a permutation of a palindrome.The palindrome does not need to be limited to just dictionary words.

This is what is wrong with my implementation:

1. It exhausts all possible options then filters identical permutations: potentially extra work recreating the original string if it is already a palindrome, extra work if the same char occurs 4,6.. times.
2. It removes spaces before deciding if a palindrome can exist => extra work
3. It is brute-force-like, not an ingenious solution with long functions, many loops, lists, and variables.
4. It only functions for strings up to 7 non-space characters.

#the space doesn't matter: place in the same location
#the first char = last char and symmetrical

#find polindromes of a string

```
def palindrome_permutation(string):
    #remember the space locations
    space_loc = []
    str_list = list(string)
    for i in range(len(str_list)):
        if str_list[i] == " ":
            space_loc.append(i)
    #get rid of spaces
    for i in range(len(space_loc)):
        str_list.remove(" ")
    #check if we could make a polindrome
    # count # of char occur
    chars = []
    num_occur = []
    new_str_size = len(str_list)
    for char in str_list:
        if char not in chars:
            chars.append(char)
            num_occur.append(1)
        else:
             num_occur[chars.index(char)]+=1
     #check if # of occur is div by 2 or ∃ only 1 char not div by 2
```

```
count = 0
center_index = None #will use to create palindrome of odd length
for index in range(len(num_occur)):
 if num_occur[index]%2 != 0:
        count+=1
        center_index = index
if count > 1:
    print("False")
    return False
#create palindromes
palindromes_list = [] #contains output

for num in range(len(num_occur)):
    num_occur[num]=int(num_occur[num]/2)

half_chars = []
for ind in range(len(num_occur)):
    for i in range(num_occur[ind]):
        half_chars.append(chars[ind])

size = len(half_chars)
operations=0
while operations<math.factorial(size):
    for i in range(size-1):
        swap(half_chars, i, i+1)
        operations+=1
        i+=1
        #local output
        palindrome = [None]*new_str_size
        #fill the center for odd length
        if center_index != None:
            palindrome[int((new_str_size)/2)] = chars[center_index]
        #insert permutation in first half of palindrome[] and #fill by symmetry
        for charind in range(size):
            palindrome[charind] = half_chars[charind]
           palindrome[new_str_size-charind-1] = palindrome[charind]
    #insert spaces and convert to string
        for s in space_loc:
            palindrome.insert(s, " ")
        palindrome = "".join(palindrome)
       #check if solution already existed
       if palindrome not in palindromes_list:
           palindromes_list.append(palindrome)
print("True, permutations: ", palindromes_list)
return True
```